Journal of Global Optimization manuscript No.
(will be inserted by the editor)

Evolutionary Annealing

Global Optimization in Measure Spaces

Alan J. Lockett · Risto Miikkulainen

Received: date / Accepted: date

Abstract Stochastic optimization methods such as evolutionary algorithms and Markov Chain Monte Carlo methods usually involve a Markov search of the optimization domain. Evolutionary annealing is an evolutionary algorithm that leverages all the information gathered by previous queries to the cost function. Evolutionary annealing can be viewed either as simulated annealing with improved sampling or as a non-Markovian selection mechanism for evolutionary algorithms. This article develops the basic algorithm and presents implementation details. Evolutionary annealing is a martingale-driven optimizer, where evaluation yields a source of increasingly refined information about the fitness function. A set of experiments with twelve standard global optimization benchmarks is performed to compare evolutionary annealing with six other stochastic optimization methods. Evolutionary annealing outperforms other methods on asymmetric, multimodal, non-separable benchmarks and exhibits strong performance on others. It is therefore a promising new approach to global optimization.

Keywords annealed selection · real-space evolutionary annealing · non-Markovian selection · evolutionary computation · martingale-driven optimization

This research was supported in part by NSF under grants DBI-0939454 and IIS-0915038.

Alan J Lockett
Department of Computer Science
University of Texas at Austin
Austin, TX, 78712
USA
Tel.: +1-512-471-9571
Fax: +1-512-471-8885
E-mail: alockett@cs.utexas.edu

Risto Miikkulainen
Department of Computer Science
University of Texas at Austin
Austin, TX, 78712
USA
Tel.: +1-512-471-9571
Fax: +1-512-471-8885
E-mail: risto@cs.utexas.edu

1 Introduction

Stochastic optimization algorithms typically operate in a Markov fashion, forgetting previously evaluated solutions. For example, in Markov Chain Monte Carlo (MCMC) methods, a single solution candidate is generated from the prior solution candidate; in evolutionary algorithms, the population for each generation is constructed stochastically from the prior population only. As a result, these algorithms can discover and then forget high-quality regions within the search domain. They can therefore fail to exploit crucial information, resulting in suboptimal performance. As shown in this paper, this problem can be alleviated by selecting individuals for reproduction over the entire pool of previously observed solutions. Although a genetic algorithm with blind non-Markovian selection can in principle become trapped by local optima, this issue can be mitigated by combining genetic algorithms and simulated annealing. The resulting algorithm, termed *evolutionary annealing* in this paper, solidly outperforms both genetic algorithms and simulated annealing and compares favorably with several state-of-the-art stochastic optimization methods.

This paper develops the evolutionary annealing approach and evaluates it experimentally. Evolutionary annealing is a global optimization algorithm for a large class of measure spaces that can be alternately viewed as a genetic algorithm with non-Markovian selection or as a method for simulated annealing without the Metropolis sampler. Evolutionary annealing introduces a new annealed selection operator, exploiting a connection between the average effect of proportional selection and the annealed Boltzmann distributions used in simulated annealing. Although many genetic algorithms have previously employed the Boltzmann distribution for selection (Goldberg, 1995; Jeong and Lee, 1996; Mühlenbein and Mahnig, 2002), evolutionary annealing is distinct from these approaches in that it can select any member of any prior population. Evolutionary annealing is distantly related to Estimation of Distribution Algorithms (EDAs), since it builds a global model of the annealing distributions for the fitness function (Pelikan et al, 2002; Mühlenbein et al, 1999). However, whereas EDAs build models based solely on the best members of the immediately prior generation, evolutionary annealing maintains a model based on the entire history of observation. By leveraging the information acquired from function evaluations, evolutionary annealing constructs an increasingly refined estimate of the fitness function that allows it to locate the global optimum. To illustrate this process, the progress of an example run of evolutionary annealing in a two-dimensional space is shown in Figure 1.

Experimentally, evolutionary annealing converges quickly on a bank of standard benchmarks. This paper includes results for eleven global optimization problems in Section 5, including several multi-modal and non-separable problems that are difficult for many optimization methods. Because of its efficient sampling and non-Markovian selection, evolutionary annealing performs well in a comparison with Simulated Annealing (SA), Differential Evolution (DE), Correlated Matrix Adaption Evolution Strategies (ES), Particle Swarm Optimization (PSO), the real-coded Bayesian Optimization Algorithm (rBOA), and a real-coded genetic algorithm (rGA). Evolutionary annealing thus deserves consideration as a potentially useful new approach to global optimization.

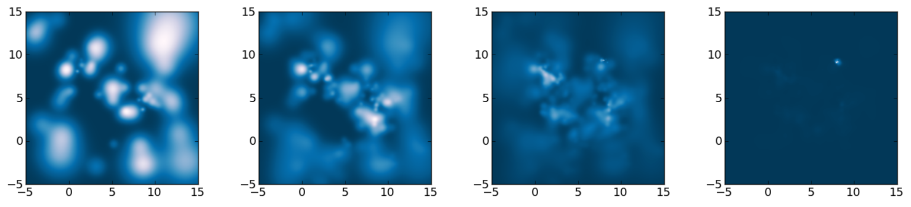


Fig. 1 Example run of evolutionary annealing on Shekel's Foxholes in two dimensions (see Figure 3(e)). Images are heat maps displaying the estimated probability density of evolutionary annealing, that is, the probability that each point will occur in the next generation of evolutionary annealing. White areas are more probable, and dark areas are less probable. Successive frames show how the probability density changes once 50, 125, 250, and 500 points have been evaluated. The resulting distribution increasingly models the fitness function; comparison with Figure 3(e) confirms that after 500 evaluations, evolutionary annealing has focused on the true global optimum.

2 Background

Before introducing evolutionary annealing, some background in simulated annealing is presented, followed by a short discussion of the theory of proportional selection in genetic algorithms that motivates the global selection mechanism of evolutionary annealing.

2.1 Simulated Annealing

Simulated annealing is an MCMC optimization algorithm that employs properties of statistical mechanics to locate minima of a given fitness function (Kirkpatrick et al, 1983; Bertsimas and Tsitsiklis, 1993). The usual analogy is that of crafting a metallic artifact by repeatedly shaping it at different temperatures. At high temperatures, the metal is malleable and easy to shape, but as such the metal does not easily remain in detailed configurations. As the temperature is gradually lowered, the overall shape become increasingly fixed in a lower-energy crystalline configuration.

At the core of the simulated annealing algorithm is the Boltzmann distribution. At time n, simulated annealing samples approximately from a distribution given by

$$\mathcal{A}_n^f\left(dx\right) = \frac{1}{Z_n}\exp\left(-\frac{f(x)}{T_n}\right)dx, \tag{1}$$

where f is the fitness function, Z_n is a normalizing factor known as the *partition function*, and T_n is a sequence of temperatures with $T_n \to 0$. The sequence T_n is known as the *cooling schedule*. The distribution $\mathcal{A}_n^f$ will be referred to as an *annealing distribution* in this paper. Simulated annealing samples from $\mathcal{A}_n^f$ repeatedly using the Metropolis algorithm (Metropolis et al, 1953; Hastings, 1970). The process begins with a proposed solution x. At each time step, a proposal distribution $\mathbb{Q}$ is used to sample x_n. The proposed solution x is replaced with x_n with probability $\exp\left(-\max\left\{0, f(x) - f(x_n)\right\}/T_n\right)$. For each fixed temperature T_n the algorithm will converge to a sample from $\mathcal{A}_n^f$. As $n \to \infty$, $\mathcal{A}_n^f$ converges in probability to a distribution that samples directly from the optimal points of f (Kirkpatrick et al, 1983).

Subject to conditions on the cooling schedule, simulated annealing can be shown to converge asymptotically to the global optima of the fitness function (Hajek, 1988; Yang, 2000). For combinatorial problems, Hajek showed that simulated annealing converges if the cooling schedule is set as $T_n \propto 1/\log n$ (Hajek, 1988). In practice, simulated annealing has been used effectively in several science and engineering problems. However, its sensitivity to the proposal distribution and the cooling schedule means that it is not a good fit for all optimization problems.

Surprisingly, traditional genetic algorithms are connected with simulated annealing through an analysis of the average performance of a genetic algorithm with proportional selection. This connection is exposed by trivial manipulations of a previous result of Mühlenbein and Mahnig (2002); the details are discussed next.

2.2 Expected Proportional Selection

Many genetic algorithms employ *proportional selection*, where individuals in the prior population are selected proportionally to their observed fitness. Much like simulated annealing, proportional selection sharpens the fitness function implicitly with each generation, so that on averaging over population trajectories the selection operator asymptotically places probability one on the optima of the fitness function. The following argument for discrete spaces is derived from Mühlenbein and Mahnig (2002); analogues to this result hold in the class of measure spaces considered here. For the purposes of this section, the goal is to maximize f, a positive fitness function.

Proportional selection at the n^{th} time step is given by $\mathcal{S}_f^n(x) \propto f(x) N_x^{n-1}$, where $\mathcal{S}_f^n(x)$ is the probability of selecting x at time n, and N_x^n is a random variable indicating the number of copies of the solution x in the population at time n. Taking the expected value over N_x^n,

$$\mathbb{E}\left[\mathcal{S}_f^n(x)\right] \propto f(x)\mathbb{E}\left[N_x^{n-1}\right]. \tag{2}$$

The expected value on the left is also a probability distribution over x and therefore a selection rule, here termed *expected proportional selection.* It is possible to imagine an evolutionary algorithm where each successive population is sampled from just this rule. This algorithm is a one-stage, selection-only genetic algorithm; because expected proportional selection averages over all individuals, no variation is required.

In such an algorithm, if the initial population is selected uniformly at random, then $\mathbb{E}\left[N_x^0\right]$ is a constant, so

$$\mathbb{E}\left[\mathcal{S}_f^1(x)\right] \propto f(x). \tag{3}$$

By definition, $\mathbb{E}[\mathcal{S}_f^n(x)] = \mathbb{E}[N_x^n]/K$ where K is the population size, since N_x^n/K is just the proportion of the population taking the value x. Applying this fact to the recursion in Equation 2 yields $\mathbb{E}[S_f^n(x)] \propto f(x)^n$. Thus expected proportional selection sharpens the fitness function. Introducing $g(x) \equiv -\log(f(x))$,

$$\begin{aligned}\mathbb{E}\left[S_f^n(x)\right] &\propto \exp\left(-g(x)\right)^n \\ &= \exp\left(-\frac{1}{n^{-1}}g(x)\right)\end{aligned} \tag{4}$$

Comparing Equation 1 to Equation 4, expected proportional selection is found to have an annealing distribution on $-\log f$ with cooling schedule $T_n = n^{-1}$. Since the logarithm is monotonic, the maxima of f are the minima of g.

Expected proportional selection is not a feasible selection rule, because it requires total knowledge of the fitness function a priori. If such knowledge were possible, there would be no need for evolutionary computation; the optima would already be known. Expected proportional selection could be estimated by averaging over the trajectories of several different runs of a genetic algorithm, but the number of trajectories required for a good estimate would be intractably large. Genetic algorithms with proportional selection can be viewed as an approximation of this selection rule.

Evolutionary annealing exploits the theoretical relationship between simulated annealing and genetic algorithms to create a hybridized algorithm that merges qualities of both algorithms, as is described next.

3 Evolutionary Annealing

This section defines the evolutionary annealing algorithm. The following notation is used throughout. Let X be a topological space with a given Hausdorff (separated) topology, and let $(X, \mathcal{F})$ be a measurable space such that $\mathcal{F}$ is the Borel σ-algebra for the given topology on X. The Borel σ-algebra is the smallest σ-algebra containing the open sets of a topology, so that open sets are always $\mathcal{F}$-measurable. Let λ be a finite measure on $(X, \mathcal{F})$ such that λ is positive on all nonempty open sets. Let $f : X \to \mathbb{R}$ be a fitness function which is to be minimized, and assume that f has all necessary integrability properties required by the formulae that follow. The notation (P_n) will represent a *stochastic population process*, that is, a sequence of populations generated by a stochastic optimization algorithm. Each population P_n contains a fixed number of individuals and is denoted by $P_n = \left(P_n^k\right)_{k=1}^K$, where K is the population size. The set A_n represents the set of all individuals up to time n, $A_n = \bigcup_{m \leq n,k}\left\{P_m^k\right\}$. With these definitions, the basic algorithm can be defined.

3.1 Basic Algorithm

Evolutionary annealing consists of selection and variation phases. The population P_{n+1} is sampled one individual at a time in these two stages. In the selection phase, an element $a \in A_n$ is selected with probability

$$p_n(a) = \xi_n^{-1} \exp\left(-\frac{f(a)}{T_n}\right)\lambda\left(E_n^a\right), \tag{5}$$

where T_n is a cooling schedule, ξ_n is a normalizing factor, and $\lambda\left(E_n^a\right)$ is the measure of a region surrounding the point a, discussed below. This selection mechanism will be termed *annealed proportional selection* based on the relationship between expected proportional selection and annealing described in the prior section. Its primary distinction as a selection rule is that it can select any member of any prior population.

For the variation phase, evolutionary annealing requires a family of probability distributions $\{\nu_n^x\}_{x\in X}$ used to mutate selected points, so that given a selected point x, ν_n^x is used to vary x at time n. The choice of mutation distributions is essentially arbitrary, although in general the variance should decrease over time to promote convergence. In Euclidean space, Gaussians can be used, centered at x and with the variation as a hyperparameter σ_n. In binary spaces, individual bits can be flipped with a probability dependent on n. The particular mutation distributions should be chosen based on the needs of the problem at hand; a mutation distribution whose shape is well matched with the shape of the fitness function will converge much faster than one that is not. Some results for a specific instantiation of evolutionary annealing with real vectors will be discussed in Section 5.

Once an individual $a \in A_n$ has been selected with probability $p_n\left(a\right)$, then that individual is mutated according to ν_n^a in order to generate a new member of the population. That is, each individual in the population at time $n+1$ is sampled according to

$$P_n^k \sim \sum_{a\in A_n} p_n(a)\nu_n^a. \tag{6}$$

Thus evolutionary annealing samples its populations from a sequence of mixture distributions with one mixing point located at each individual from prior populations. In this way, the selection is non-Markovian; the selected individual could come from any previous generation. The mixture probabilities $p_n(a)$ are chosen according to the annealing formula in Equation 5.

Intuitively, if the temperature is fixed at a constant, as the number of mixing points increases and the variance of the mutation distribution decreases, the mixture distribution in Equation 6 approximates the annealing distribution $\mathcal{A}_n^f$ in Equation 1. It is commonly known that mixtures of Gaussians can model any sufficiently smooth distribution arbitrarily well if enough mixing points are used. It is also true that mixture distributions in general can model any probability measure arbitrarily well subject to certain conditions. The population P_n is successively sampled from better and better approximations to $\mathcal{A}_n^f$, and as $n \to \infty$, the population sequence P_n will increasingly focus on the optima of f.

A high-level algorithm for evolutionary annealing over N generations is shown in Algorithm 1. The algorithm depends on two subroutines, *prepare* and *sample*. The subroutine *prepare* builds data structures to support efficient sampling of the quantity p_n from Equation 5. The subroutine *sample* samples from p_n using the prepared data structures. Through the use of highly precise approximations as described in Section 3.3, both *prepare* and *sample* can be implemented to run in time logarithmic in the population size and the number of generations. The specific implementations of *prepare* and *sample* used in the experiments utilize the methods of Section 3.3. The *prepare* routine adds nodes to the trees described in that section and propagates the components of Equations 7 and 14 up the tree. The *sample* routine employs Equations 7 and 14 to traverse the tree down from

Algorithm 1 Evolutionary Annealing Algorithm

```
$N$, the number of generations
$K$, sample points (population size) per generation
$\left(P_1^k\right)_{k=1}^{K}$, the initial random population
$A_0 \leftarrow \emptyset$, all points from all generations
for $n \leftarrow 1$ to $N$ do
    $A_n \leftarrow \bigcup_k P_n^k \cup A_{n-1}$
    $p_n \leftarrow \mathit{prepare}\left(A_n\right)$
    for $k \leftarrow 1$ to $K$ do
        $y \leftarrow \mathit{sample}\left(p_n\right)$
        $P_{n+1}^k \leftarrow$ a sample from $\nu_n^y$
    end for
end for
```

the root in order to select a previously evaluated point. Assuming that sampling ν_n^a and computing $\lambda\left(E_n^a\right)$ do not add to the complexity, the overall algorithm has performance $O\left(NK \log NK\right)$.

In order to make evolutionary annealing concrete, the cooling schedule must be determined. In light of Hajek (1988), a default choice for the cooling schedule is given by $T_n^{-1} = \eta \log n$. Here η is a learning rate that scales the fitness function and thereby controls the aggressiveness of selection. A high learning rate focuses selection on the few best individuals and may restrict exploration of the space. A low learning rate allows promiscuous selection, slowing down refinement of previously discovered solutions but increasing the probability of escaping a local minimum. Again following Hajek (1988), a possible value for η is $1/d$ where d is the largest depth of a local minima relative to its surroundings in the fitness landscape. In more complex spaces, different cooling schedules could be considered. There may also be a benefit to linking the variance of the mutation distribution to the cooling schedule, so that as the probability of selecting the current best individual decreases, the variance also decreases to enable refined exploration of the immediate region around the current best.

The region weight $\lambda\left(E_n^a\right)$ is present in Equation 5 to avoid a particular scenario of premature convergence. Once a good solution is discovered, evolutionary annealing will devote increasing resources to exploring the neighborhood of that point. If these points are also good, then the probability of selecting more points in the same region will increase in a feedback loop. Within a few generations, almost all points selected will come from the immediate environment of these good points. If there is a local minimum in the vicinity, evolutionary annealing would likely become entrapped in that region. The region weight is intended to serve as a measure of how many individuals have been previously sampled in the region surrounding the point a. The sets E_n^a partition X around points in A_n, the total population so far. Such a partition can be computed in logarithmic time in many spaces. These partitions may also be thought of as a source of increasingly refined information about the fitness function, evoking the concept of martingales as studied in stochastic process theory.

Algorithm 2 Algorithm to Generate a Partition Based On Grid Points

```
$\{x_m\}_{m=1}^M \subseteq X$, the mixing points
$\mathcal{T} \leftarrow \{X\}$, the partition tree
$k(i) \leftarrow \emptyset$ for all $i = 1, \ldots, M$, node assignment function
for $m \leftarrow 1$ to $M$ do
  $N \leftarrow$ the leaf node in $\mathcal{T}$ such that $x_m \in N$
  if $\exists j \neq m$ s.t. $k(j) = N$ then
    $N_0, N_1 \leftarrow \mathit{separate}\,(x_j, x_m, N)$
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{N_0, N_1\}$
    $k(j) \leftarrow N_0$, $k(m) \leftarrow N_1$
  else
    $k(m) \leftarrow N$
  end if
end for
```

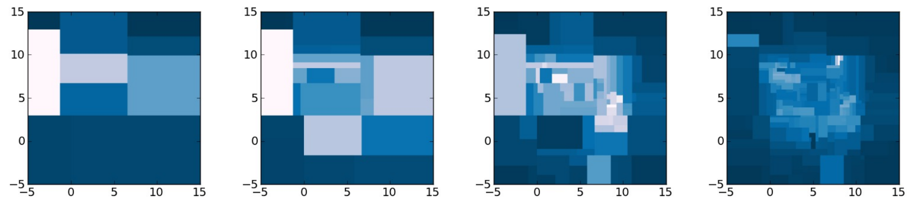
3.2 Partitioning the Space

Each of the mixing points $a \in A_n$ will be considered representative of a particular region of the search space X. Each successive set A_n will be associated with a partition $\{E_n^a\}_{a \in A_n}$ of disjoint sets such that $X = \bigcup_{a \in A_n} E_n^a$ and $a \in E_n^a$ for all n. The σ-algebra $\mathcal{F}$ is assumed to be rich enough to support such partitions based on any finite collection of points in X. The partitioning set E_n^a is the same as the one that appears in Equation 5.

Provided that there exists a computable algorithm to split any set containing two distinct points into two disjoint sets each of which contains exactly one of the points, then the partitions can be stored in a binary tree, and if the splitting algorithm does not depend on the population size of the number of generations, the computational complexity of maintaining a partitioning tree is logarithmic on average.

Algorithm 2 partitions any Borel measure space over a Hausdorff topology given a function for dividing a partition region between two separate points in the region. A partition is represented as a binary tree, with the root representing the entire space X and each branch partitioning X into two sets. The algorithm is initialized with a sequence of points $\{x_m\}_{m=0}^M \subseteq X$ to be partitioned (the mixing points), a tree $\mathcal{T}$ with X as the root node, and an assignment function k such that $k(m)$ is the leaf node of the tree assigned to the point x_m, or $\emptyset$ if no assignment has been made. The algorithm then loops through the mixing points, splitting the space where necessary to ensure that each leaf node contains exactly one mixing point. The algorithm relies on a domain-specific subroutine to split an existing set, *separate*. At the end of each iteration of the algorithm's main loop, each leaf node is assigned to exactly one mixing point. When a new mixing point is added, *separate* partitions the leaf node to which it belongs into two new leaf nodes, each containing only one mixing point. The process of adding a single new mixing point to the tree requires only a tree traversal, so that at each generation, updating the partition requires $O(K \log NK)$ time, where NK is the number of points at the N^{th} generation.

In a vector space, such as $\mathbb{R}^d$, the function *separate* can in many cases be given explicitly. Suppose that X is bounded above by $\{u_i\}$ and below by $\{\ell_i\}$ so that X has a rectangular shape. Each node in the partition tree will restrict the coefficient for exactly one of the basis vectors, say j. To maintain computability, it is necessary



(a) 10 points (b) 25 points (c) 100 points (d) 250 points

Fig. 2 Progression of partition regions $\{E_n^a\}$ on Shekel's Foxholes during the run of evolutionary annealing in Figure 1 in two dimensions. Images are heat maps displaying the selection probability of each region; light regions have a higher selection probability. Successive frames show how the partition regions gradually model the shape of the fitness function after 10, 25, 100, and 250 points have been evaluated.

to require that $j < D < \infty$ for some D. That is, each set E_n^a in the partition is defined as a hyperrectangle on finitely many coordinates, with each step in the traversal of the partitioning tree adding a new coordinate value for some side of the hyperrectangle. So E_n^a can be represented as two vectors, u^a for the upper bounds, and ℓ^a for the lower bounds. Given the point $a \in X$ and a second point $x \in X$, E_n^a can be separated as follows. Let $k = \text{argmax}_{i \leq D} |a_i - x_i|$; k is the index at which the rectangle E_n^a will be split. Suppose $a_k > x_k$ for the sake of simplicity; the opposite situation is handled analogously. Initialize $u^x \leftarrow u^a$ and $\ell^x \leftarrow \ell^a$. Then set $\ell_k^a \leftarrow \frac{1}{2}(a_k + x_k)$ and $u_k^x \leftarrow \frac{1}{2}(a_k + x_k)$. The regions E_{n+1}^a and E_{n+1}^x defined by these boundary vectors are then disjoint if the upper boundary is strict. The result of this partitioning method in $\mathbb{R}^2$ is shown in Figure 2. This version of *separate* cannot separate two vectors that are the same in the first D coefficients. In an infinite-dimensional vector space, it is possible for two distinct vectors to have arbitrarily many identical coefficients, and no computable algorithm and locate the coefficients in which they differ. This situation is of theoretical more than practical concern, however, and can be ignored in most cases. The separation algorithm above can be efficiently implemented in many spaces of interest. The next section discusses how these partition mechanisms can be used to implement the subroutines *prepare* and *sample* from Algorithm 1.

3.3 Sampling Algorithms for Annealed Selection

The computational efficiency of evolutionary annealing is primarily determined by the cost of preparing and sampling annealed proportional selection. A naïve approach to computing Equation 5 would make the cost of preparing and sampling p_n be linear, since the normalizing factor ξ_n must be computed one element at a time and because sampling from a probability vector typically requires iterating through the vector. In fact, annealed proportional selection can be approximately computed in logarithmic time in the average case by leveraging the partition tree, with most operations occurring in subroutines that guarantee worst-case logarithmic time, to be described in Section 3.4. The approximation can be made accurate at close to machine-level precision, so that it is sufficiently precise for all practical purposes.

In order to reduce the sampling complexity for evolutionary annealing from linear to logarithmic time, a tree-sampling method is needed for sampling p_n. The partition tree provides a tree such that the leaves are associated exactly with the components of p_n. The goal, then, is to create a sequence of decisions made along a path through the partition tree such that the decision process assigns probability mass to each complete path in equality with the probability of the leaf at the end of the path under p_n.

Let ν be an internal node of the partition tree. Let $N \subseteq A_n$ be the set of previously observed individuals residing within leaves of the partition tree that are descended from ν. Let μ be one of the two child nodes of ν, and let $M \subseteq N$ contain the leaf descendants of μ. To extend a probabilistic path that has reached ν, a choice must be made at node ν whether to add node μ or its sibling to the path. Suppose the choice is made according to

$$\mathbb{P}\left(\mu \mid \nu\right) = \frac{\sum_{x\in N} \alpha(x)^{\log n} \lambda\left(E_n^x\right)}{\sum_{y \in M} \alpha(y)^{\log n} \lambda\left(E_n^y\right)}, \tag{7}$$

where $\alpha(x) \equiv \exp\left(-\eta f(x)\right)$, mirroring Equation 5 with cooling schedule $T_n^{-1} = \eta \log n$. Now let π_x be a path from the root to the leaf containing the point x, and observe that a sequence of decisions made according to Equation 7 yields

$$\mathbb{P}\left(\pi_x\right) = \prod_{\nu} \mathbb{P}\left(\text{child}\left(\nu, \pi_x\right) \mid \nu\right) = \xi_n^{-1} \alpha(x)^{\log n} \lambda\left(E_n^x\right) = p_n\left(x\right), \tag{8}$$

with child (ν, π_x) being the child node of ν on the path π_x. The next to last equality in Equation 8 holds because each successive denominator cancels the numerator of the previous one, leaving only the denominator from the root node, which is equal to ξ_n, and the numerator from the leaf node, which is $\alpha(x)^{\log n} \lambda\left(E_n^x\right) = \exp\left(-f(x)/T_n\right) \lambda\left(E_n^x\right)$. Therefore, sampling a path through the tree starting from the root samples from p_n provided that the decision at each node is made according to Equation 7.

The difficulty of this method is that the sum in the numerator of Equation 7 must be computed for each node. If the temperature were fixed, then the value of the sum could be stored on each node. The sum only changes when new leaves are inserted, and then only the nodes that are direct ancestors of the inserted node need to adjust their sums, resulting in logarithmic updates to the tree. As long as the temperature does not change, then, the tree-sampling method is logarithmic both to prepare the data structures and to sample them.

It remains to account for changes in temperature without recomputing the numerator of Equation 7 at each time step. Introducing $h(T) = \sum_{x \in N} \alpha(x)^T \lambda\left(E_n^x\right)$ to capture the fact that the sum varies with the generation, the problem is that the exponent cannot be pulled out of the sum, meaning that the sum must be recomputed with every change in temperature. However, $h(T)$ is infinitely differentiable in T, with m^{th} derivative

$$h^{(m)}(T) = \sum_{x \in N} \alpha(x)^T \left(\log \alpha(x)\right)^m \lambda\left(E_n^x\right). \tag{9}$$

Thus a Taylor approximation is possible, since

$$h(T) = \sum_{m=1}^{\infty} \left(\sum_{x \in N} \frac{\left(\log \alpha(x)\right)^m}{m!} \alpha(x)^{T_0} \lambda\left(E_n^x\right)\right) \left(T - T_0\right)^m. \tag{10}$$

The Taylor approximation can be computed by storing a vector of coefficients $t = (t_1 \dots t_m)$ with $t_j \equiv \sum_{x \in N} (\log \alpha(x))^j \alpha(x)^{T_0} \lambda(E_n^x)$ for all $j \in 1 \dots m$, with a fixed value T_0. These vector sums can then be propagated up the tree in logarithmic time, and the sampling method can approximate $h(\log n)$ as needed at each node.

To complete the description of the sampling method, T_0 and m must be specified. As a general feature of $h(T)$, the approximation is substantially correct for $T > T_0$ over a larger interval than for $T < T_0$. With $m = 10$, the approximation is highly accurate for $T \in [T_0, T_0 + 1/2]$ but degrades outside that interval. Thus the Taylor coefficients must be recomputed for the entire tree on every interval of T of size $1/2$. For practical purposes, the value of T_0 is set to 1 for the first few generations, and then is reset when $T = \log n = 3/2, 2, 5/2, \dots$. This resetting feature is actually not as burdensome as it may sound, and it only needs to be performed logarithmically often, so that the entire procedure of maintaining and sampling the tree still has logarithmic complexity overall. Some example statistics for computation time are shown in Table 1. The next section discusses an alternate annealed selection rule, analogous to tournament selection, and introduces data structures that make it possible to sample annealed selection in average case logarithmic time.

3.4 Annealed Tournament Selection

Annealed proportional selection as given in Equation 5 is a proportional selection rule; individuals are selected according to their proportion of the overall fitness. Proportional selection has a well-known drawback that also applies to annealed proportional selection. For example, suppose that the fitness function f has a minimal value of 0, and consider the selection probabilities for the points x, y with $f(x) = 0.01$ and $f(y) = 0.001$ at temperature $T_n = 5$. Assume $\lambda(E_n^x) = \lambda(E_n^y) = 1$. Then $p_n(y)/p_n(x) = 1.0018$. That is, x is almost equally as likely to be selected as y, even though y is a whole order of magnitude closer to the optimum. Thus the more precise solution is no more likely to be selected than rougher solutions close to the optimum, which makes refinement of solutions near a local or global optimum sluggish. These intuitions are confirmed by the experimental results in Table 4; annealed proportional selection converges within 0.1 of the optimal fitness without difficulty, but then fails to attain accuracy within 0.001 in most cases.

To address this weakness of proportional selection in genetic algorithms, tournament and ranking selection were introduced (cf. Syswerda (1989)). These methods select among individuals according to their fitness rank in the population rather than according to their raw fitness. For tournament selection, the best individual is selected with some probability q, termed the *selection pressure*. If the best individual is not selected, then the second best individual is chosen with probability q. Thus the probability of selecting the n^{th}-ranked individual of the population is proportional to $q(1-q)^{n-1}$.

A similar concept can be used to define *annealed tournament selection*, a non-Markovian version of tournament selection. Annealed tournament selection replaces Equation 5 by

$$p_n(a) = \xi_n^{-1} \, q^{1/T_n} \left(1 - q^{1/T_n}\right)^{r(a)} \lambda(E_n^a), \tag{11}$$

where q is the selection pressure, and $r(a)$ is the fitness rank of a in A_n starting with 0. Annealed tournament selection uses a cooling schedule T_n so that the rank becomes increasingly significant with each generation, with the ultimate result that the top-ranked individual is selected at zero temperature. The main difference from standard tournament selection is that each individual must be ranked against all other individuals from all prior generations. As a consequence, the selection pressure must be much lower. For this paper, the value of q was fixed at 0.025. Rather than varying q, the learning rate η in the cooling schedule can be varied to achieve the same effect.

As with annealed proportional selection, it is not computationally efficient to sample Equation 11 directly. In addition, annealed tournament selection introduces the need to sort all previously proposed solutions by fitness. In order to accommodate these issues, a balanced binary tree can be used, called the *score tree*. Like the partition tree, the score tree contains one leaf node per proposed solution; the internal nodes represent the set of nodes in their span. The score tree reorganizes the partition tree so that points with higher fitness are always to the left and points with lower fitness are always to the right. Using standard tree algorithms, the score tree can be balanced in logarithmic time after each insertion.

Annealed ranking selection can be sampled by walking the score tree, making a decision at each node whether to follow the lower- or the higher-ranked branch. The probability at each node will depend on the area represented by the node and the height of the subtree underneath the node. The area of a leaf node can be copied from the partition tree. Both the area and the height can then be propagated up the score tree in logarithmic time after each insertion. In this way, the score tree is also a partition tree. However, the internal nodes of the score tree correspond approximately to the level sets of the fitness function, and thus the regions that they represent can be arbitrarily complex to describe. Therefore, although the score tree defines a partition over the search space, the score tree cannot replace the partition tree, because there is no efficient way to determine whether a point resides in the region represented by an internal node of the score tree. However, the score tree is kept balanced, providing worst-case logarithmic performance.

When sampling annealed tournament selection using the score tree, the decision must be made at each internal node ν whether to follow the higher- or lower-ranked branch. Let $h+1$ be the height of the subtree under node ν, and assume the tree is perfectly balanced. Then ν has 2^{h+1} leaf nodes in its span. Let μ be the higher-ranked child node of ν. Suppose further that the nodes spanned by ν range in rank from R to $R+2^{h+1}-1$, so that the nodes spanned by μ range in rank from R to $R+2^h-1$. Ignoring the region weight temporarily, a direct application of standard tournament selection yields

$$\mathbb{Q}_T\left(\mu \mid \nu\right) = \frac{\sum_{m=0}^{2^h-1} q^{1/T}\left(1-q^{1/T}\right)^{R+m}}{\sum_{j=0}^{2^{h+1}-1} q^{1/T}\left(1-q^{1/T}\right)^{R+j}}. \tag{12}$$

Let κ be the lower ranked sibling of μ, spanning ranks $R+2^h$ to $R+2^{h+1}-1$. Then the ratio for selecting μ over κ is given by

$$\frac{\mathbb{Q}_T\left(\mu \mid \nu\right)}{\mathbb{Q}_T\left(\kappa \mid \nu\right)} = \frac{\sum_{m=0}^{2^h-1} q^{1/T}\left(1-q^{1/T}\right)^{R+m}}{\sum_{m=0}^{2^h-1} q^{1/T}\left(1-q^{1/T}\right)^{R+2^h+m}} = \frac{1}{\left(1-q^{1/T}\right)^{2^h}} \equiv \tilde{q}\left(h,T\right). \tag{13}$$

The function $\tilde{q}(h, T)$ gives the selection preference of the higher branch over the lower branch. Finally, incorporating the region weights, let

$$\mathbb{P}_T\left(\mu \mid \nu\right) = \frac{\tilde{q}\left(h, T\right)\lambda\left(\mu\right)}{\tilde{q}\left(h, T\right)\lambda\left(\mu\right) + \left(1 - \tilde{q}\left(h, T\right)\right)\lambda\left(\kappa\right)}, \tag{14}$$

where $\lambda\left(\mu\right)$ and $\lambda\left(\kappa\right)$ are the cumulative weights of the partition regions of the points in the span of μ and κ, respectively. Equation 14 is normalized and implies $\mathbb{P}_T\left(\kappa \mid \nu\right) = 1 - \mathbb{P}_T\left(\mu \mid \nu\right)$.

To show that this process does in fact implement annealed tournament selection, notice that

$$\mathbb{P}_T\left(\mu \mid \nu\right) \propto \tilde{q}\left(h, T\right)\frac{\lambda\left(\mu\right)}{\lambda\left(\nu\right)} \;\; , \;\; \mathbb{P}_T\left(\kappa \mid \nu\right) \propto \frac{\lambda\left(\mu\right)}{\lambda\left(\nu\right)}, \tag{15}$$

introducing the $\lambda(\nu)$ factor as a proportional constant. Thus for a general path π_x, recalling that $\tilde{q}(h, T) \propto \mathbb{Q}_T\left(\mu \mid \nu\right)$ by definition,

$$\mathbb{P}_T\left(\text{child}\left(\nu, \pi_x\right) \mid \nu\right) \propto \mathbb{Q}_T\left(\text{child}\left(\nu, \pi_x\right) \mid \nu\right)\frac{\lambda\left(\mu\right)}{\lambda\left(\nu\right)} \tag{16}$$

and therefore

$$\begin{aligned}
\mathbb{P}_{T_n}\left(\pi_x\right) &= \prod_{\nu \in \pi_x} \mathbb{P}_{T_n}\left(\text{child}\left(\nu, \pi_x\right) \mid \nu\right) \\
&\propto \prod_{\nu \in \pi_x} \mathbb{Q}_{T_n}\left(\text{child}\left(\nu, \pi_x\right) \mid \nu\right)\frac{\lambda\left(\text{child}\left(\nu, \pi_x\right)\right)}{\lambda\left(\nu\right)} \\
&= \mathbb{Q}_{T_n}\left(\pi_x\right)\frac{\lambda\left(E_n^x\right)}{\lambda\left(X\right)} \\
&\propto p_n\left(x\right).
\end{aligned} \tag{17}$$

The last equality holds because the area ratios successively cancel each other, and the last proportionality follows from the fact that $\mathbb{Q}_{T_n}$ was defined to implement tournament selection with selection pressure q^{1/T_n}. The ultimate conclusion is that a tree-sampling algorithm with node selection probabilities as given in Equation 14 can be used to sample from annealed tournament selection in worst-case logarithmic time.

As a final note on efficiency, notice that sampling in the score tree has worst-case logarithmic time, whereas sampling on the partition tree has average case logarithmic time. Therefore it makes sense to sample annealed proportional selection from the score tree rather than the partition tree. The only additional requirement is that the Taylor coefficients for annealed proportional selection should be propagated up the score tree rather than the partition tree. In this way, regardless of whether tournament or proportional selection is used, the sampling operations of evolutionary annealing require logarithmic time in the worst case.

Table 1 Performance statistics for Evolutionary Annealing on a 2GHz Intel Core 2 Duo processor using the open-source implementation. For each number of observed points, the table gives the time in milliseconds for sampling one point, for inserting one point into the partition tree, for inserting one point into the ranked score tree, and for the total processing overhead per function evaluation. Complexity grows logarithmically in the number of points.

points	sample	partition	rank	total
1,000	8.6	18.2	20.6	59.2
5,000	10.5	22.1	24.7	64.5
10,000	11.2	24.1	26.4	68.1
25,000	11.8	27.6	28.2	76.8
50,000	12.4	34.0	30.4	99.2
100,000	12.9	47.3	32.8	113.6

3.5 Implementation

Because evolutionary annealing relies on several data structure, it can be complex to implement. In order to further clarify implementation details and to permit the reproducibility of the experimental results that follow, an open-source implementation has been released under the name *pyec* (http://pypi.python.org/pypi/PyEC). This package implements both annealed proportional and tournament selection along with many other popular evolutionary computation methods, including the exact code used to run the experiments described in Section 5. This package is intended to encourage further experimentation and evaluation of the evolutionary annealing method beyond the results reported in this paper.

Performance statistics for evolutionary annealing were gathered using this implementation in order to demonstrate the actual computational costs of running the algorithm in Table 1. These statistics were compiled by averaging results from four runs each of the algorithm using tournament selection on the benchmarks *shekel* and *rastrigin*. Tournament and proportional selection both traverse the score tree when sampling, so the numbers are representative for both selection rules. The columns of Table 1 show the average time required for sampling the score tree, for inserting a point into the partition tree, for inserting a point into the ranked score tree, and for the total processing overhead per individual. Each entry shows the average time in milliseconds to process a single individual given a certain number of stored points in the database. The averages are cumulative, so for example the fact that sampling requires 12.9 ms with $100,000$ points in the database means that the average sample time over all $100,000$ individuals was 12.9 ms. As an exception, the total processing time per individual shows the cost per individual averaged over 100 samples. Logarithmic growth in complexity is clear from the table.

3.6 Martingale-Driven Optimization

From an intuitive perspective, evolutionary annealing performs successfully because it builds an increasingly accurate model of the fitness function. In this sense, evolutionary annealing shares certain general concepts in common with metamodelling techniques (El-Beltagy et al, 1999), such as those using Kriging methods (Jeong et al, 2005). For evolutionary annealing, progressive partitions of the search domain serve as a source of increasing information. In the language of martingale theory, the sequence of partitions generates a *filtration.* If the temperature and the mutation distributions are held constant, it is conjectured that under certain smoothness assumptions, the random sequence of fitness values generated by evolutionary annealing is an approximate martingale. It may also be possible to extend such martingale-like properties to non-constant cooling schedules in order to prove asymptotic global convergence in some cases. For these reasons, evolutionary annealing may be described as a martingale-driven optimizer. This martingale characterization also motivated the development of the partitioning approach used by evolutionary annealing. The approach therefore offers an opportunity to develop an interesting new optimization theory in the future.

Now that the algorithm has been fully described, its experimental performance will be explored next.

4 Experimental Setup

Evolutionary annealing can be used to search for bit strings, real vectors, neural networks, Bayesian network structures, game strategies, programs, state machines, and any other structure that can be embedded within a suitable measure space. To verify the algorithm, experiments were performed in real-vector space on a set of twelve standard benchmarks. The instantiation of evolutionary annealing in Euclidean space is termed Real-Space Evolutionary Annealing (REA). REA was tested with both annealed proportional selection and annealed tournament selection; these two variants are termed REA-P and REA-T, respectively. To clarify, the experiments were performed in a hypercube $Q \subseteq \mathbb{R}^d$, with a normalized Lebesgue measure on the Borel σ-algebra restricted to Q, i.e. $\lambda(B) = \int_B dx / \int_Q dx$.

The version of REA implemented for this article uses Gaussian mutation distributions with $\nu_n^a = \mathcal{N}\left(a, \sigma_n\left(a\right)^2\right)$. The standard deviation $\sigma_n(a)$ is scaled to the area of the partition region with $\sigma_n(a) = \frac{1}{2} w \lambda\left(E_n^a\right)^{1/d}$, where d is the dimension of the problem and w is the width of the space (i.e. $\frac{1}{2}w$ is the side length of Q). This choice of variance seeks to align the shape of ν_n^a and $\lambda\left(E_n^a\right)$. Specifically, if E_n^a were a hypercube, then the first standard deviation of ν_n^a would be contained within E_n^a.

In high dimensions, the variance of the mutation distributions needs to be forcibly decayed. Otherwise, the algorithm will not focus for an exponentially long time. In the experiments that follow, no decay factor was applied for $d = 5$ and $d = 10$. It was not necessary to do so, since the samples in the experiments converged towards a fixed distribution without a decay factor. In 25 dimensions, however, a decay factor of $n^{-\frac{1}{2}}$ (i.e. $\sigma_n(a) = \frac{1}{2} w n^{-\frac{1}{2}} \lambda\left(E_n^a\right)^{1/d}$) was applied in order to achieve faster convergence.

This implementation of REA differs from that of Lockett and Miikkulainen (2011b) in that it uses actual rather than approximated values for the area. It dif-

Table 2 Benchmarks for Experimental Validation, with $d = 5$

Name	Definition	Minimum	Domain
sphere	$\sum_{i=1}^{d} x_i^2$	0.0000	(-5.12 , 5.12)
ackley	$-20\exp(-\frac{.02}{d}\|\|x\|\|^2) - \exp(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)) + 20 + e$	0.0000	(-30 , 30)
log-ackley	$\sum_{i=1}^{d-1} e^{-0.2}\sqrt{x_i^2 + x_{i+1}^2} + 3\cos(2x_i) + 3\sin(2x_{i+1})$	-13.3796	(-30 , 30)
whitley	$\sum_{i=1}^{d}\sum_{j=1}^{d}\frac{w(x_i,x_j)^2}{4000} - \cos(w(x_i,x_j)) + 1$, with $w(y,z) = 100\left(y^2 - z\right)^2 + (1-z)^2$	0.0000	(-30 , 30)
shekel	$\sum_{i=1}^{30}\frac{1}{\sum_{j=1}^{d}(x_j - a_{ij})^2 - c_i}$	-10.4056	(-5 , 15)
rosenbrock	$\sum_{i=1}^{d-1} 100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2$	0.0000	(-5.12 , 5.12)
rastrigin	$10d + \sum_{i=1}^{d} x_i^2 - 10\cos(2\pi x_i)$	0.0000	(-5.12 , 5.12)
salomon	$-\cos(2\pi\|x\|) + 0.1\|x\| + 1,\ \|x\| \equiv \left(\sum_i x_i^2\right)^{1/2}$	0.0000	(-30 , 30)
langerman	$-\sum_{i=1}^{5} c_i \exp(-y_i/\pi)\cos(\pi y_i),\ y_i = \sum_{j=1}^{d}(x_j - a_{ij})^2$	-0.9650	(-5 , 15)
schwefel	$d^{-1}\sum_{i=0}^{d} -x_i \sin\sqrt{\|x_i\|}$	-418.9829	(-512 , 512)
griewank	$1 + \sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_i \cos(x_i/\sqrt{i})$	0.0000	(-600 , 600)
weierstrass	$\sum_{i=1}^{d}\sum_{j=1}^{20} 0.5^j \cos\left(2\cdot 3^j \pi(x_i + 0.5)\right) + d\sum_{j=1}^{20} 0.5^j \cos\left(3^j\pi\right)$	0.0000	(-0.5 , 0.5)

fers from that of Lockett and Miikkulainen (2011a) in that the variance is scaled to the area rather being controlled by a decay schedule. These improvements account for the superior results shown below.

The twelve benchmarks are defined in Table 2. These benchmarks are commonly used and cover a broad cross-section of possible fitness functions; their definitions and descriptions can be found in the literature (e.g. de Jong (1975); Ali et al (2005); Mühlenbein et al (1991); Whitley et al (2003); Bersini et al (1996); Ackley (1987)). Notably, two versions of Ackley's function exist; both are included in the comparisons. The less common one is termed *log-ackley* and is due to Ackley (1987). The more common version of the benchmark is exponentiated and centered and is simply termed *ackley*. Each benchmark was tested in five, ten, and 25 dimensions ($d = 5, 10, 25$), except that *shekel* and *langerman* were tested in five and ten dimensions only, since they are not defined in 25 dimensions. The domain for each benchmark was a bounded hypercube with the range for each component shown in the table. The minima for these functions are known, as shown for five dimensions with precision up to 10^{-4}. The actual values are known up to machine-level precision (10^{-16}), and these more accurate values were used for testing the accuracy of the experiments. Heat maps of the eleven benchmarks with $d = 2$ are shown in Figure 3.

In order to establish the performance of REA relative to other optimization algorithms, experiments were run with six other algorithms: (1) simulated annealing (SA), (2) a real-coded genetic algorithm (rGA), (3) an evolution strategy (CMA-ES), (4) differential evolution (DE), (5) particle swarm optimization (PSO), and (6) the real-coded Bayesian Optimization Algorithm (rBOA). These algorithms cover a broad spectrum of stochastic optimization algorithms and represent a general sampling of the current state of the art. They are known to be effective on a wide array of fitness functions and most of them perform reasonably well on the selected benchmarks. For all of the algorithms, parameters were set according to the literature where available and hand-tuned otherwise to optimize performance.

CMA-ES is the Correlated Matrix Adaption algorithm of Hansen and Ostermeier (2001) and was tested with four different population sizes: 100, 750, 1250, and 2500. At each generation, 50% of the population was used to build an updated normal distribution. Because CMA-ES converges quickly, CMA-ES was restarted whenever the covariance matrix collapsed to an average variance of $1e - 25$ (see e.g. Auger and Hansen (2005)).

DE was trained with four different parameter settings, one each with crossover rates 0.2 and 0.9 and learning rates .2 and .9 (Storn and Price (1995)). PSO were trained with both the global and local adaptation rates set to 2.0 (Eberhart and Kennedy (1995)). The velocity decay was tested with two different values -0.5 and 1.0 following results by Pedersen (2010) on optimal parameter settings for PSO. In particular, rBOA is an Estimation of Distribution Algorithm (EDA), and was included to compare REA's performance with other sampling-based optimizers. It was implemented as described in Ahn et al (2006).

Simulated annealing was run as a single chain with a logarithmic cooling schedule. It was restarted randomly with probability 0.001 after each point. The rGA method was a standard real-coded genetic algorithm using linear ranking selection with pressure 1.8, uniform crossover, and Gaussian mutation. The mutation variance for rGA was set to 0.05 for all problems except *schwefel* and *griewank*, where it was set to 10.

The parameters for REA-P and REA-T are the learning rate η and the population size K. Several values for η were tested, shown in Table 5 for each benchmark. Preliminary experiments showed that the learning rate influences the performance of REA more than the population size; thus experiments varying the population size were left for future work. REA-P was not tested in 25 dimensions to conserve computational resources; preliminary experiments showed that REA-T substantially outperformed REA-P in 25 dimensions, as it does in five and ten dimensions.

All algorithms were run on all benchmarks 200 times for each tested parameter setting. These 200 runs are sufficient to guarantee statistical significance on the estimated success rates for each algorithm at the 95% level within $+/-$ 0.5% (Vasile et al (2011)). When a single number is shown as the result of an experiment, that number represents the best value achieved on any parameter setting for that algorithm, unless otherwise stated.

5 Experimental Results

The complete experimental results are given in Tables 4 to 9. Tables 4, 5, and 6 show the success rates in 5, 10, and 25 dimensions, respectively, over all trials for each algorithm and benchmark at three different error levels: 0.1, 0.01, and 0.001 in five and ten dimensions, and 10, 1, and 0.1 for 25 dimensions. The success rate for an error level ϵ is calculated as the percentage of trials in which the algorithm achieved an error less than ϵ from the global optimum. Tables 7, 8, and 9 give the average error across all trials (successful and unsuccessful) in 5, 10, and 25 dimensions after $10,000$, $100,000$, and $250,000$ evaluations.

In short, REA-T, DE, and CMA-ES are the most effective algorithms on this set of benchmarks. REA-T is more effective on problems that are asymmetric, non-separable, and multimodal such as *shekel*, *langerman*, and *whitley.* DE outperforms REA-T on some but not all radially symmetric problems such as *rastrigin*, *salomon*, and *griewank.* CMA-ES performs well, particularly on *rastrigin* and *griewank*, but the effectiveness of CMA-ES is likely due to the restart mechanism, which could also be used to boost the performance of REA or DE. Comparing the two versions of REA, REA-P performs well, but fails to refine solutions near global and local optima. In contrast, REA-T attains precisely refined solutions, most often at the global optimum, and is therefore the stronger method on these benchmarks.

More specifically, in five dimensions, the results show clearly that REA-P and REA-T are effective at locating the global optima of complex fitness functions. REA-P is successful on most problems at the 0.1 success level, with notable exceptions for *rastrigin* and *schwefel.* For *schwefel*, REA-P actually located the region of the true global optimum on most trials, but was unable to refine these solutions further. For comparison, the failures of CMA-ES and PSO on this benchmark were over an order of magnitude worse and were not in the correct region of the search space. On *rastrigin*, it was not possible to configure REA-P to succeed predictably. It is possible that the algorithm would succeed at a lower learning rate (e.g. $\eta = 0.001$) with more function evaluations, but an even lower learning rate would further slow down the refinement of the solution.

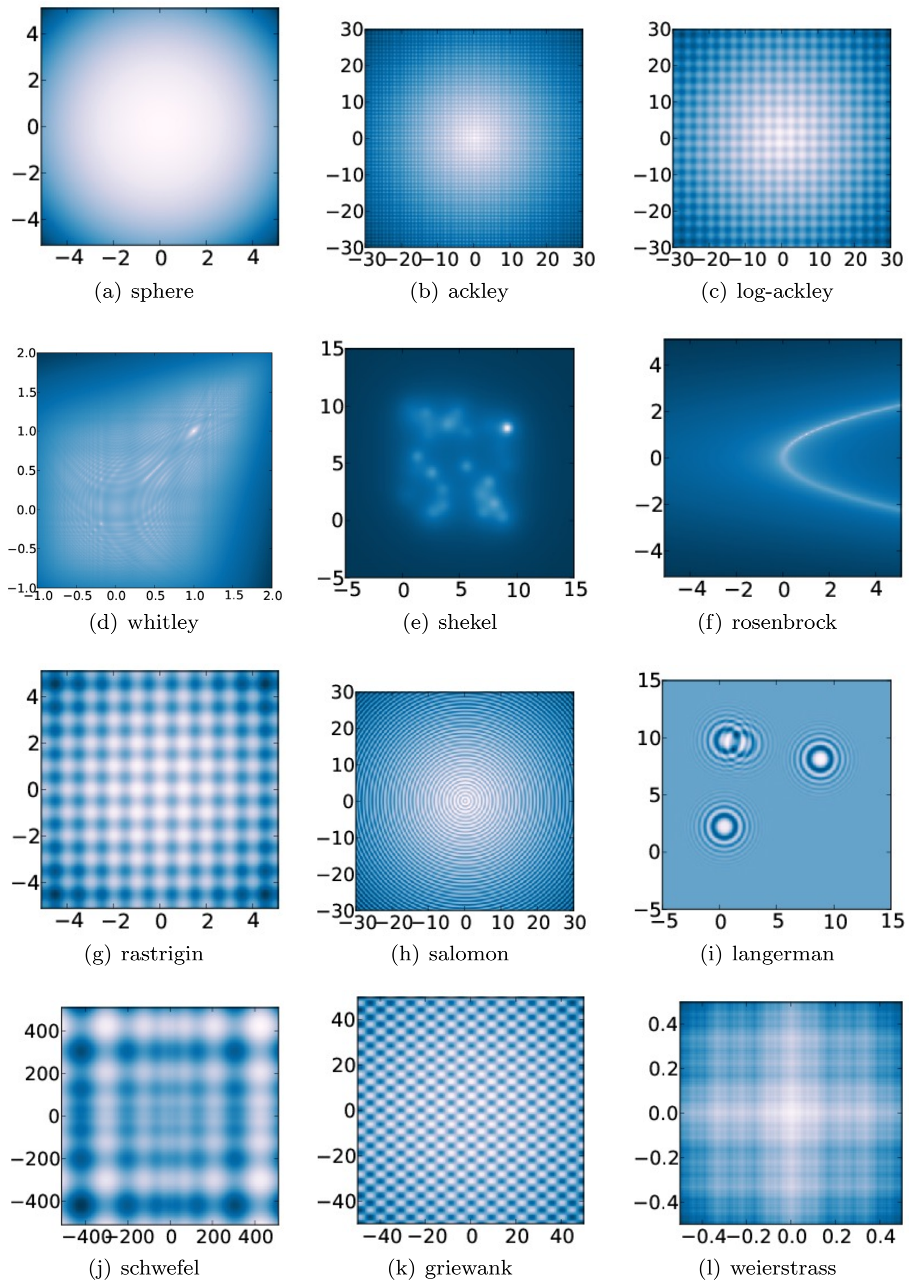
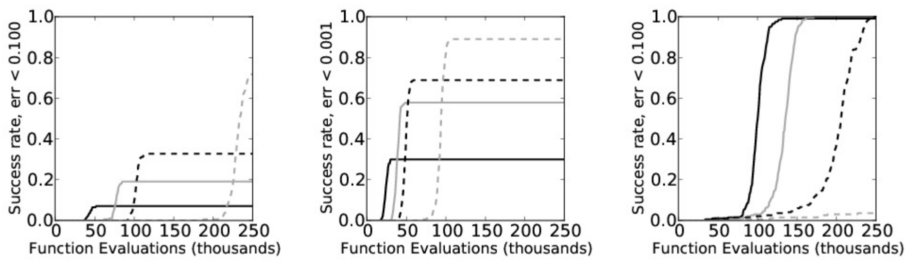


Fig. 3 Heat maps for the eleven benchmark functions in two dimensions ($d = 2$). Experiments were performed in five dimensions ($d = 5$). The benchmarks *whitley* and *griewank* are scaled to show the critical region. Lighter colors indicate lower and therefore more optimal values. These three functions are multimodal and non-separable and are quite difficult for most optimization methods. This set of benchmarks contains a variety of difficult problems that should reveal distinctions among optimization algorithms.

Table 3 Learning rates η for REA-P and REA-T tested in the experiments.

	REA-P		REA-T		
Benchmark	$d = 5$	$d = 10$	$d = 5$	$d = 10$	$d = 25$
sphere	10	1, 10	10	1, 10	10
ackley	0.25	0.25, 1	0.05, 0.25	0.25, 1	0.5
log-ackley	0.25	0.25, 1	0.25	0.25, 1	0.5
whitley	0.1	0.25, 1	0.05, 0.25	0.25, 1	25
shekel	0.1, 0.25	0.1, 1	0.1, 0.5, 1.0, 5.0	0.1, 1	–
rosenbrock	1	1, 5	5	1, 5	10
rastrigin	0.01, 0.1	0.035, 1	0.01, 0.035, 0.050, 0.075	0.035, 1	1
salomon	2	1, 2	2	1, 2	5
langerman	0.1, 0.5	0.25, 1	0.1, 0.5, 1.0, 5.0	0.25, 1	–
schwefel	0.015	0.01, 0.001	0.001	0.01, 0.001	0.005
griewank	1, 10	0.1, 1	0.025, 0.1, 0.25, 0.5	0.1, 1	1
weierstrass	5	1, 5	5	1, 5	10



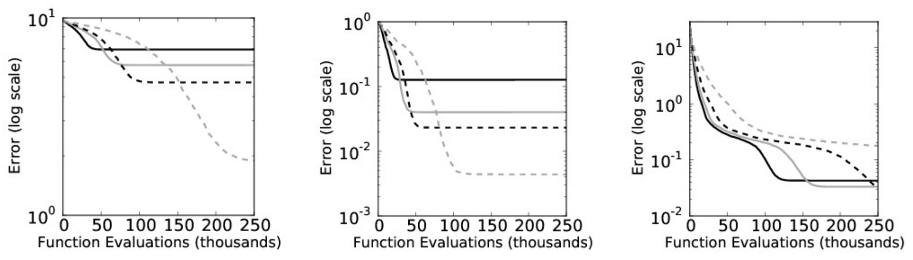
(a) shekel (b) langerman (c) griewank

Fig. 4 Success probabilities for REA-T on selected benchmarks in five dimensions for four different learning rates. Decreasing the learning rate improves the success probability overall but requires more fitness evaluations.

By contrast, REA-T is very effective at refining points around the optima. In most cases where REA-T came within 0.1 of the optima, it also managed to attain machine-level precision. The exceptions to this statement primarily involved local optima with fitness values close to those of the true optimum (i.e. *salomon*, *langerman* and *griewank*). In the case of *rastrigin*, tournament selection even helped REA-T escape local optima in several cases, so that it attained the true global optimum more often than REA-P.

In higher dimensions, all of the algorithms had trouble attaining the global optimum. However, a review of the errors in Table 8 shows that REA-T was competitive with the others. In preliminary trials, REA-P failed on *whitley* and *rosenbrock* is a consequence of numeric issues. In both of these problems, the region of the search space containing reasonable fitness values (e.g. $f(x) < 100$) is small relative to the overall area, and in higher dimensions this region becomes exponentially smaller. Annealed proportional selection overflows on large fitness values (Equation 5) and must therefore be capped, so the probability that REA-P selects any particular point is effectively constant. This can be overcome by using a very small learning rate, but then REA-P would not be able to converge once the feasible region is attained. Because annealed tournament selection is only sensitive to the fitness rank of points, REA-T does not suffer from numeric issues and continues to perform relatively well on *whitley* and *rosenbrock* even in higher dimensions. It is possible that with lower learning rates, REA-T could perform even better in 25 dimensions.

Figure 4 shows the progression of the success probability and Figure 5 the magnitude of the error as a function of the number of evaluations for REA-T with different learning rates on selected benchmarks. As the learning rate is decreased, REA-T converges slower and succeeds more often. Thus there is a trade-off between the number of evaluations and solution quality. A higher learning rate can be used to reduce the number of evaluations, but at the cost of reducing the probability of success. In Figure 4, notice that the shape of the graph remains remarkably constant while the learning rate changes, suggesting that the success probability changes smoothly and predictably as a function of the learning rate and the number of evaluations.



(a) shekel (b) langerman (c) griewank

Fig. 5 Average error rates for REA-T on selected benchmarks in five dimensions for four different learning rates. The black solid line is the average error for the largest learning rate in Table 5. The grey solid line is the second largest learning rate. The black dotted line is the third largest, and the grey dotted line is the smallest learning rate. Decreasing the learning rate reduces error overall at the cost of increased error in early generations.

6 Discussion and Future Work

The experimental results in Section 5 are favorable for evolutionary annealing, especially with annealed tournament selection. There are some generalizations that may be drawn from the results. First, REA-T is generally better than REA-P for optimization and is thus the preferred implementation for Euclidean space. Second, REA is most successful relative to other algorithms on problems that do not possess an easily identifiable structure, such as *langerman* and especially *shekel*. The reason is that REA does not assume a particular problem structure in its definition. In structured domains, such as *sphere*, REA may use more function evaluations than would otherwise be necessary to eliminate the possibility that the current best solution is a local optimum. However, in unstructured environments, these extra function evaluations help REA avoid becoming trapped in local optima.

Another interesting result is the good performance of DE. DE is an elegant and simple algorithm and is consequently more computationally efficient than REA-T, performing up to two orders of magnitude faster in terms of per-generation overhead. However, in real-world problems, the computation of fitness values typically far outweighs the cost of algorithmic overhead. The overhead of REA is generally unrelated to the fitness function being optimized, so in domains where the fitness takes a long time to compute, the use of REA will not add substantially to the overall computation time.

Also, the results on the benchmarks suggest that DE and REA-T are complementary, with REA-T being preferable on highly unstructured problems, and DE performing better on problems with some degree of symmetry around the optimum. In practice, there are many real-world problems both with and without symmetry. If the degree of structure is not known, and fitness can be calculated quickly, a reasonable approach is to test DE first and use REA-T if DE fails.

CMA-ES also performed well, mostly due to the restarts. Without restarts, it is not comparable to either DE or REA-T. Restarting after convergence is a form of boot-strapping that can augment the probability of success. For example, if an algorithm has a 5% chance of success but converges after $1,000$ evaluations, then

Table 4 Percentage of successful trials at various error levels for REA and six other algorithms on the benchmark set in five dimensions.

error <	sphere	ackley	log-ackley	whitley	shekel	rosenbrock	rastrigin	salomon	langerman	schwefel	griewank	weierstrass
REA-P												
0.100	1.00	0.98	1.00	1.00	0.00	0.99	0.29	1.00	0.01	0.26	0.69	0.90
0.010	1.00	0.05	0.57	1.00	0.00	0.99	0.01	0.00	0.00	0.00	0.02	0.00
0.001	1.00	0.00	0.00	1.00	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00
REA-T												
0.100	1.00	0.29	0.99	1.00	0.73	1.00	0.39	1.00	1.00	1.00	1.00	0.96
0.010	1.00	0.29	0.99	1.00	0.73	1.00	0.39	0.00	0.89	1.00	0.33	0.89
0.001	1.00	0.29	0.99	1.00	0.73	1.00	0.39	0.00	0.89	1.00	0.05	0.85
DE												
0.100	1.00	1.00	0.93	0.18	0.11	0.42	0.92	1.00	0.31	0.79	1.00	0.49
0.010	1.00	0.77	0.93	0.16	0.05	0.12	0.78	0.00	0.13	0.74	0.18	0.25
0.001	1.00	0.01	0.93	0.15	0.04	0.01	0.65	0.00	0.01	0.71	0.01	0.19
CMA-ES												
0.100	1.00	1.00	1.00	0.61	0.00	0.13	1.00	1.00	1.00	0.39	1.00	1.00
0.010	1.00	1.00	1.00	0.39	0.00	0.02	1.00	0.00	1.00	0.38	1.00	1.00
0.001	1.00	1.00	1.00	0.15	0.00	0.01	1.00	0.00	1.00	0.37	1.00	0.00
PSO												
0.100	0.82	0.00	0.00	0.06	0.00	0.00	0.01	0.07	0.00	0.11	0.00	1.00
0.010	0.16	0.00	0.00	0.05	0.00	0.00	0.00	0.01	0.00	0.11	0.00	1.00
0.001	0.02	0.00	0.00	0.04	0.00	0.00	0.00	0.01	0.00	0.11	0.00	1.00
rGA												
0.100	1.00	0.00	0.02	0.69	0.02	0.07	0.01	1.00	0.50	0.52	0.07	0.00
0.010	1.00	0.00	0.02	0.26	0.00	0.00	0.00	0.00	0.25	0.01	0.00	0.00
0.001	1.00	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.24	0.00	0.00	0.00
rBOA												
0.100	1.00	1.00	0.97	0.00	0.00	0.00	1.00	1.00	0.10	0.00	1.00	0.00
0.010	1.00	1.00	0.01	0.00	0.00	0.00	1.00	0.86	0.00	0.00	1.00	0.00
0.001	1.00	1.00	0.00	0.00	0.00	0.00	1.00	0.85	0.00	0.00	1.00	0.00
SA												
0.100	1.00	0.00	0.00	0.00	0.00	1.00	0.12	0.19	0.13	0.00	0.00	0.00
0.010	1.00	0.00	0.00	0.00	0.00	0.12	0.00	0.00	0.00	0.00	0.00	0.00
0.001	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 5 Percentage of successful trials at various error levels for REA and six other algorithms on the benchmark set in in ten dimensions.

error <	sphere	ackley	log-ackley	whitley	shekel	rosenbrock	rastrigin	salomon	langerman	schwefel	griewank	weierstrass
REA-P												
0.100	1.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.010	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.001	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
REA-T												
0.100	1.00	0.00	0.17	0.12	0.13	0.00	0.00	1.00	0.64	0.00	0.30	1.00
0.010	1.00	0.00	0.17	0.10	0.13	0.00	0.00	0.00	0.35	0.00	0.00	1.00
0.001	1.00	0.00	0.17	0.07	0.13	0.00	0.00	0.00	0.24	0.00	0.00	1.00
DE												
0.100	1.00	0.58	0.44	0.06	0.03	0.02	0.13	0.17	0.00	0.24	0.95	0.29
0.010	1.00	0.00	0.44	0.04	0.03	0.01	0.11	0.00	0.00	0.22	0.01	0.07
0.001	1.00	0.00	0.44	0.04	0.03	0.00	0.07	0.00	0.00	0.21	0.00	0.01
CMA-ES												
0.100	1.00	1.00	1.00	0.00	0.00	0.00	1.00	1.00	0.60	0.00	1.00	0.27
0.010	1.00	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.43	0.00	1.00	0.00
0.001	1.00	1.00	1.00	0.00	0.00	0.00	1.00	0.00	0.23	0.00	1.00	0.00
PSO												
0.100	0.46	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
0.010	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
0.001	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
rGA												
0.100	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00
0.010	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.001	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
rBOA												
0.100	1.00	1.00	0.00	0.00	0.00	0.00	0.98	1.00	0.00	0.00	1.00	0.00
0.010	1.00	1.00	0.00	0.00	0.00	0.00	0.96	0.01	0.00	0.00	1.00	0.00
0.001	1.00	0.95	0.00	0.00	0.00	0.00	0.93	0.01	0.00	0.00	1.00	0.00
SA												
0.100	1.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
0.010	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.001	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 6 Percentage of successful trials at various error levels for REA-T and six other algorithms on the benchmark set in in 25 dimensions. Experiments were not performed on REA-P. Results marked with "–" are not available.

error <	sphere	ackley	log-ackley	whitley	shekel	rosenbrock	rastrigin	salomon	langerman	schwefel	griewank	weierstrass
REA-T												
10.000	1.00	1.00	0.26	0.00	–	0.00	0.00	1.00	–	0.00	1.00	0.55
1.000	1.00	1.00	0.00	0.00	–	0.00	0.00	1.00	–	0.00	1.00	0.00
0.100	1.00	0.00	0.00	0.00	–	0.00	0.00	0.00	–	0.00	1.00	0.00
DE												
10.000	1.00	1.00	0.71	0.01	–	0.01	0.03	1.00	–	0.00	1.00	0.00
1.000	1.00	1.00	0.01	0.00	–	0.00	0.00	0.98	–	0.00	0.97	0.00
0.100	1.00	0.00	0.01	0.00	–	0.00	0.00	0.00	–	0.00	0.18	0.00
CMA-ES												
10.000	1.00	1.00	1.00	0.00	–	0.00	1.00	1.00	–	0.00	1.00	0.08
1.000	1.00	1.00	0.76	0.00	–	0.00	0.98	1.00	–	0.00	1.00	0.00
0.100	1.00	1.00	0.04	0.00	–	0.00	0.34	0.02	–	0.00	1.00	0.00
PSO												
10.000	1.00	1.00	0.00	0.00	–	0.00	0.06	1.00	–	0.00	1.00	1.00
1.000	0.87	0.00	0.00	0.00	–	0.00	0.00	0.76	–	0.00	0.00	1.00
0.100	0.11	0.00	0.00	0.00	–	0.00	0.00	0.00	–	0.00	0.00	1.00
rGA												
10.000	1.00	1.00	0.00	0.00	–	0.00	0.00	1.00	–	0.00	1.00	0.00
1.000	1.00	0.00	0.00	0.00	–	0.00	0.00	1.00	–	0.00	0.00	0.00
0.100	0.01	0.00	0.00	0.00	–	0.00	0.00	0.00	–	0.00	0.00	0.00
rBOA												
10.000	1.00	1.00	0.00	0.00	–	0.00	0.00	1.00	–	0.00	–	–
1.000	1.00	0.14	0.00	0.00	–	0.00	0.00	1.00	–	0.00	–	–
0.100	1.00	0.00	0.00	0.00	–	0.00	0.00	0.05	–	0.00	–	–
SA												
10.000	1.00	1.00	0.00	0.00	–	0.04	0.00	1.00	–	0.00	1.00	0.00
1.000	1.00	0.00	0.00	0.00	–	0.00	0.00	0.12	–	0.00	0.00	0.00
0.100	0.01	0.00	0.00	0.00	–	0.00	0.00	0.00	–	0.00	0.00	0.00

Table 7 Average error from the global optimum for REA and six other algorithms on the benchmark set in five dimensions.

# evals	sphere	ackley	log-ackley	whitley	shekel	rosenbrock	rastrigin	salomon	langerman	schwefel	griewank	weierstrass
REA-P												
10K	0.0014	0.6645	0.2860	4.1455	9.1972	0.3840	2.9208	0.1411	0.7238	15.3999	0.0948	0.2544
100K	0.0003	0.1904	0.0187	0.0009	8.6825	0.1635	0.9406	0.0999	0.5216	8.1417	0.0854	0.1751
250K	0.0002	0.0297	0.0093	0.0002	8.1382	0.0416	0.8270	0.0999	0.4629	7.2595	0.0852	0.1634
REA-T												
10K	0.0000	0.3658	0.1585	0.5154	4.7938	0.1080	1.9750	0.1004	0.0061	94.6310	0.1090	0.0332
100K	0.0000	0.1571	0.0148	0.0000	1.8679	0.0008	0.6400	0.1004	0.0044	0.8934	0.0167	0.0332
250K	0.0000	0.1571	0.0148	0.0000	1.8679	0.0000	0.6400	0.1004	0.0044	0.1148	0.0167	0.0332
DE												
10K	0.0327	0.2301	3.9922	14.0266	8.2814	7.2069	5.5317	0.3458	0.5413	51.1648	0.8970	2.3108
100K	0.0000	0.0273	0.0748	0.8001	6.3685	0.4990	0.3057	0.1010	0.3312	3.7434	0.0362	0.4428
250K	0.0000	0.0081	0.0504	0.4958	5.3834	0.2082	0.0904	0.0999	0.1934	0.5712	0.0192	0.1304
CMA-ES												
10K	0.0000	0.0057	0.0746	2.5028	7.7216	1.5900	0.1274	0.0999	0.1120	79.7161	0.0039	0.0752
100K	0.0000	0.0000	0.0000	0.6328	7.7010	0.6603	0.0000	0.0979	0.0008	15.5293	0.0000	0.0069
250K	0.0000	0.0000	0.0000	0.3650	7.6696	0.3813	0.0000	0.0967	0.0001	3.4695	0.0000	0.0053
PSO												
10K	0.3721	0.8254	15.9946	23.9864	9.3539	52.0516	22.0823	0.7562	0.8812	54.4219	2.3898	0.0000
100K	0.1134	0.8254	11.7666	17.2840	9.3014	12.9145	13.2249	0.3740	0.8074	31.7351	1.3701	0.0000
250K	0.0560	0.8254	9.4647	14.5696	9.2475	8.0872	7.8817	0.2488	0.7464	28.3746	1.1319	0.0000
rGA												
10K	0.0008	1.0661	6.6267	4.9425	7.9254	3.2157	3.6592	0.1156	0.8261	13.6210	0.2996	1.9318
100K	0.0003	1.0600	6.6217	1.5089	7.9221	0.5365	3.5530	0.0999	0.4343	13.4689	0.1715	1.4962
250K	0.0002	1.0591	6.6210	0.6611	7.9216	0.3548	3.5366	0.0999	0.2907	13.4421	0.1456	1.3239
rBOA												
10K	0.0000	0.0039	0.3573	9.2064	8.6679	2.7962	0.0325	0.0861	0.5325	202.7886	0.0019	4.1167
100K	0.0000	0.0000	0.0838	7.7386	8.2771	1.9153	0.0000	0.0296	0.3389	202.7886	0.0000	3.0470
250K	0.0000	0.0000	0.0553	7.2232	8.1515	1.7363	0.0000	0.0118	0.2346	202.7886	0.0000	2.7295
SA												
10K	0.0129	2.2398	12.4181	15.8667	9.2187	0.7032	3.7314	0.4835	0.5921	88.1220	0.6510	2.5142
100K	0.0036	1.6883	7.6468	12.1221	8.6723	0.0413	0.9623	0.1571	0.3567	38.7923	0.4007	1.6260
250K	0.0024	1.4796	6.0146	10.7741	8.3998	0.0213	0.4407	0.1165	0.2534	26.5272	0.3362	1.3987

Table 8 Average error from the global optimum for REA and six other algorithms on the benchmark set in ten dimensions.

# evals	sphere	ackley	log-ackley	whitley	shekel	rosenbrock	rastrigin	salomon	langerman	schwefel	griewank	weierstrass
REA-P												
10K	1.2816	2.9332	39.2224	10290.6042	9.9212	2026.8125	53.3078	1.1971	0.9647	182.6232	8.0640	3.8439
100K	0.0127	1.7823	4.3236	10278.3484	9.8904	2023.3608	30.7584	0.3743	0.9625	101.6657	1.0660	1.0079
250K	0.0066	1.4815	2.7809	10274.3850	9.8792	2023.2741	16.4958	0.2655	0.9599	51.7409	0.9186	0.7805
REA-T												
10K	1.1747	2.6459	48.4113	217.8259	9.8480	209.7775	50.9295	1.1824	0.9391	191.4335	9.1123	0.3833
100K	0.0000	1.1121	10.9317	59.4486	7.6282	4.8343	28.7889	0.0999	0.3332	128.9061	0.6628	0.0000
250K	0.0000	0.5163	1.4483	37.5053	7.5466	1.4308	5.7291	0.0999	0.1578	61.0893	0.1944	0.0000
DE												
10K	0.9683	0.5597	30.5945	120.2473	9.7916	141.6437	31.8115	1.0215	0.9571	124.8040	4.2583	7.9924
100K	0.0000	0.2442	0.8536	19.5469	8.4152	6.3449	5.0540	0.2769	0.7952	29.0542	0.1533	2.3122
250K	0.0000	0.1090	0.8287	9.8597	8.2945	4.4068	1.4784	0.1844	0.6892	8.9315	0.0550	0.4861
CMA-ES												
10K	0.0000	0.0814	0.8008	43.7258	8.7291	4.5322	1.2388	0.1746	0.5331	149.1124	0.0000	1.1718
100K	0.0000	0.0000	0.0144	29.3378	8.7290	2.9451	0.0923	0.0999	0.2036	67.8206	0.0000	0.1949
250K	0.0000	0.0000	0.0048	21.7652	8.7290	2.5906	0.0050	0.0999	0.0848	44.7819	0.0000	0.1359
PSO												
10K	3.1563	1.3015	56.1784	829.7286	9.8715	580.9521	68.5763	1.6269	0.9649	102.9768	11.0369	0.0000
100K	0.4342	1.3015	32.0125	96.1409	9.8447	48.8349	28.8361	0.7732	0.9642	68.2257	2.4714	0.0000
250K	0.1762	1.3015	24.0507	82.0301	9.8410	24.7544	16.6018	0.5232	0.9633	57.0167	1.5404	0.0000
rGA												
10K	0.0189	1.6778	24.0532	64.2035	9.4948	42.6395	14.4449	0.4252	0.9650	35.6409	1.1269	7.0541
100K	0.0081	1.6366	23.9707	18.4345	8.6296	6.8330	13.0390	0.2300	0.9568	34.6184	0.9480	6.2102
250K	0.0067	1.6290	23.9597	12.6458	8.6262	6.3802	12.7188	0.2035	0.9495	34.4023	0.8853	5.9259
rBOA												
10K	0.0000	0.5366	7.2174	45.1846	9.8539	8.3404	14.8954	0.1064	0.9586	263.3000	0.0934	12.9621
100K	0.0000	0.0044	1.9313	39.7107	9.7943	7.1137	0.1660	0.0991	0.8849	263.3000	0.0000	11.4815
250K	0.0000	0.0003	1.4484	38.2696	9.7567	6.7238	0.0091	0.0989	0.8338	263.3000	0.0000	10.9342
SA												
10K	0.1053	3.6105	49.2222	90.7408	9.9228	7.2152	25.7667	1.6150	0.9623	181.8532	1.4520	7.8582
100K	0.0290	2.7737	34.6730	81.3822	9.8884	1.1786	11.0039	0.4501	0.8636	139.4907	1.2528	5.6153
250K	0.0204	2.5157	30.7139	78.0091	9.8790	0.9954	8.7575	0.3218	0.7483	122.4533	1.1963	4.9173

Table 9 Average error from the global optimum for REA-T and six other algorithms on the benchmark set in 25 dimensions. Experiments were not performed on REA-P. Results marked with "–" are not available.

# evals	sphere	ackley	log-ackley	whitley	shekel	rosenbrock	rastrigin	salomon	langerman	schwefel	griewank	weierstrass
REA-T												
10K	0.6830	2.2356	99.9259	572.5708	–	121.9265	148.3260	0.8928	–	249.6619	5.9184	11.6565
100K	0.0000	0.7536	13.8660	308.3136	–	21.4421	33.8284	0.3334	–	120.2600	0.1156	9.6998
250K	0.0000	0.4562	13.5260	307.0071	–	18.8879	33.7987	0.3334	–	114.7795	0.0155	9.6997
DE												
10K	13.1568	1.0705	173.4037	$> 1,000$	–	$> 1,000$	167.4945	2.8946	–	215.4525	163.9754	34.0136
100K	0.0233	0.6902	12.7121	394.1946	–	62.8874	58.1930	1.1432	–	119.5052	9.6759	24.8876
250K	0.0000	0.5623	8.0549	200.9167	–	28.3837	22.5758	0.6939	–	52.4583	0.3757	21.2950
CMA-ES												
10K	0.0000	0.1765	5.5256	307.0426	–	21.6387	7.8055	0.2290	–	213.3997	0.0000	–
100K	0.0000	0.0183	0.9967	274.7629	–	19.9623	0.4094	0.1937	–	148.1339	0.0000	–
250K	0.0000	0.0151	0.4486	269.2735	–	19.6717	0.1444	0.1832	–	127.6947	0.0000	–
PSO												
10K	16.1643	1.6411	182.6610	$> 1,000$	–	$> 1,000$	215.7511	2.8994	–	114.2552	52.5740	0.0000
100K	1.5605	1.6411	91.1958	648.0338	–	169.9239	70.1407	1.1598	–	111.5886	5.9244	0.0000
250K	0.5531	1.6411	69.5376	583.0651	–	78.1241	40.1857	0.7641	–	111.3858	2.8514	0.0000
rGA												
10K	2.7338	2.7491	111.8603	$> 1,000$	–	831.5931	91.2257	1.9477	–	86.4422	5.0470	27.2776
100K	0.1618	2.6254	106.6524	440.9746	–	43.4134	77.0771	0.8040	–	81.2136	2.5967	25.7650
250K	0.1485	2.5691	106.5397	396.8235	–	37.0972	68.2562	0.7685	–	80.1761	2.4768	25.2179
rBOA												
10K	8.8785	1.7451	71.3647	$> 1,000$	–	$> 1,000$	192.3534	0.7421	–	318.6973	–	–
100K	0.0302	1.3043	45.5708	427.2415	–	25.8229	117.9322	0.2353	–	318.6973	–	–
250K	0.0005	1.1479	37.8726	359.7293	–	23.1964	97.8854	0.1575	–	318.6973	–	–
SA												
10K	0.8579	5.1015	203.8520	691.8408	–	32.6484	158.3882	4.6431	–	269.7318	3.7479	29.0008
100K	0.2314	4.2868	152.1270	619.7807	–	21.7801	90.0144	1.8432	–	243.7339	2.7312	21.4549
250K	0.1700	3.9081	138.1078	607.2996	–	16.4160	78.0113	1.1724	–	231.8363	2.4612	18.8158

by running the algorithm 100 times, that 5% success rate can be boosted to 99.4%. CMA-ES converges quickly, and thus benefits from numerous restarts. If the learning rate for η is set at a high level (e.g. > 1), then REA-T will converge quickly as well. If this convergence can be measured, then REA-T could be restarted to boost its success rate. Such an extension is an interesting direction for future work.

In contrast to DE and CMA-ES, evolutionary annealing is well-defined in any suitable measure space, and its convergence results apply to integrable fitness functions (rather than just real functions, as for DE). Thus evolutionary annealing can be used to search for neural networks, game strategies, Bayesian network structure and many other problem domains where it is unclear how DE might be applied. In fact, preliminary experiments have been performed in all these problem domains with promising results (Lockett and Miikkulainen, 2011b; Lockett, 2012).

The benchmark set also shows that REA performs well on problems to which it should not be particularly well-suited, at least while using Gaussian variation. For instance, separable problems such as *schwefel* and *weierstrass* can be more efficiently solved by searching in only one dimension. For instance, rGA succeeds on *schwefel* by using recombination to cross-pollinate correct components, and DE by sharing component-level information among the different members of its population through its unique crossover mechanism. In contrast, REA must learn each component separately. While this aspect of REA could be improved for *schwefel* by implementing a mutation distribution that employs crossover, it is nonetheless promising that REA is able to learn the correct value for all components independently without using excessively more function evaluations than the other algorithms.

Given that REA-T is designed to search a space exhaustively for the global optimum, it might be expected to perform worse than more greedy algorithms in high-dimensional spaces. The results show that the opposite is true: REA-T is still one of the best algorithms in 25 dimensions. One reason is the addition of the decay factor $n^{-\frac{1}{2}}$; without this decay factor, REA-T failed to find good solutions in 25 dimensions. To see why, consider that in d dimensions, 2^d evaluations must be performed in order to cut the average side length of a partition region E_n^a in half. Thus the variance $\sigma_n(a)$ reduces exponentially slowly in high dimensions. The decay factor forces evolutionary annealing to focus only on the most promising solutions. In this way, evolutionary annealing can obtain good solutions in reasonable time in high dimensions at the cost of global optimality.

Future experimentation on evolutionary annealing will focus on training complex structures such as neural networks and game strategies. The purpose of defining evolutionary annealing at such a high level of abstraction in this paper is to provide a means for developing new algorithms to search in high-level spaces without having to reinvent the underlying evolutionary apparatus from whole cloth. The description in this article provides heuristics for setting learning parameters for a wide variety of search domains.

On the theoretical side, as was briefly suggested in Section 3.6, it might be possible to use martingale arguments to prove that evolutionary annealing converges asymptotically to the global optimum in some cases. It is not clear at this time what restrictions on the algorithm or the fitness function would be required to prove such convergence. Based on preliminary efforts, such a result must carefully coordinate both the rate of cooling and the mutation variance in conjunction with the local fluctuations of the fitness function. These efforts will hopefully yield convergence theorems for specific classes of fitness functions, which in turn will help clarify the best candidates for optimization by evolutionary annealing.

More work also remains to be done to establish the rate of convergence for evolutionary annealing. For example, maximum likelihood estimates of mixture distributions with increasing mixing points are known to approximate continuous distributions at a relatively fast rate of $C\left(\frac{\log n}{n}\right)^{0.25}$ (Genovese and Wasserman, 2000). The distributions employed in evolutionary annealing are not the same, but similar performance may be possible on continuous fitness functions.

Ultimately, the success of evolutionary annealing will be measured in terms of real-world applications. It is difficult to predict in advance whether evolutionary annealing will be successful in this effort, and which applications are the best. However, the results on the benchmarks in this paper make it clear that evolutionary annealing is worthy of consideration as a method for global optimization in general-purpose domains.

7 Conclusion

Evolutionary annealing leverages shared aspects of simulated annealing and genetic algorithms resulting in a new algorithm that is experimentally reliable and also amenable to mathematical analysis. Evolutionary annealing was implemented for real vectors and shown to compare favorably with several standard methods in several benchmark problems. Evolutionary annealing is therefore a promising new direction for experimental and theoretical research in global optimization.

Acknowledgements The authors would like to thank the anonymous reviewers for their helpful comments and advice. This research was supported in part by the NSF under grants DBI-0939454 and IIS-0915038.

References

Ackley DH (1987) A connectionist machine for genetic hillclimbing. Kluwer Academic Publishers, Norwell, MA, USA

Ahn C, Ramakrishna R, Goldberg D (2006) Real-coded bayesian optimization algorithm. In: Lozano J, Larrañaga P, Inza I, Bengoetxea E (eds) Towards a New Evolutionary Computation, Studies in Fuzziness and Soft Computing, vol 192, Springer Berlin / Heidelberg, pp 51–73

Ali MM, Khompatraporn C, Zabinsky ZB (2005) A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. Journal of Global Optimization 31:635–672, URL http://dx.doi.org/10.1007/s10898-004-9972-2, 10.1007/s10898-004-9972-2

Auger A, Hansen N (2005) A restart cma evolution strategy with increasing population size. In: Evolutionary Computation, 2005. The 2005 IEEE Congress on, pp 1769–1776

Bersini H, Dorigo M, Langerman S, Seront G, Gambardella LM (1996) Results of the first international contest on evolutionary optimisation (1st iceo). In: Proceedings of IEEE International Conference on Evolutionary Computation, pp 611–615

Bertsimas D, Tsitsiklis J (1993) Simulated annealing. Statistical Science 8(1):10–15

Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan, pp 39–43

El-Beltagy M, Nair PB, Keane AJ (1999) Metamodeling techniques for evolutionary optimization of computationally expensive problems: Promises and limitations. In: GECCO'99, pp 196–203

Genovese C, Wasserman L (2000) Rates of convergence for the gaussian mixture sieve. Annals of Statistics 28(4):1105–1127

Goldberg DE (1995) A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. Complex Systems 4:445–460

Hajek B (1988) Cooling schedules for optimal annealing. Mathematics of Operation Research 13(4):311–329

Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9(2):159–195

Hastings W (1970) Monte carlo sampling methods using markov chains and their applications. Biometrika 57(1):97–109

Jeong I, Lee J (1996) Adaptive simulated annealing genetic algorithm for system identification. Engineering Applications of Artificial Intelligence 9(5):523 – 532

Jeong S, Murayama M, Yamamoto K (2005) Efficient optimization design method using kriging model. Journal of Aircraft 42(2):413–420

de Jong KA (1975) An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680

Lockett A, Miikkulainen R (2011a) Measure-theoretic evolutionary annealing. In: Proceedings of the 2011 IEEE Congress on Evolutionary Computation (CEC-2011)

Lockett A, Miikkulainen R (2011b) Real-space evolutionary annealing. In: Proceedings of the 2011 Genetic and Evolutionary Computation Conference (GECCO-2011)

Lockett AJ (2012) General-purpose optimization through information maximization. PhD thesis, University of Texas at Austin

Metropolis N, Rosenbluth A, Rosenbluth M, Teller A, Teller E (1953) Equations of state calculations by fast computing machines. Journal of Chemical Physics 21(6):1087–1092

Mühlenbein H, Mahnig T (2002) Mathematical analysis of evolutionary algorithms. In: Essays and Surveys in Metaheuristics, Operations Research/Computer Science Interface Series, Kluwer Academic Publisher, pp 525–556

Mühlenbein H, Schomisch M, Born J (1991) The parallel genetic algorithm as function optimizer. Parallel Computing 17:619–632

Mühlenbein H, Mahnig T, Rodriguez AO (1999) Schemata, distributions, and graphical models in evolutionary optimization. Journal of Heuristics 5:215–247

Pedersen M (2010) Tuning & simplifying heuristical optimization. PhD thesis, University of Southampton

Pelikan M, Goldberg D, Lobo F (2002) A survey of optimization by building and using probabilistic models. Computational Optimization and Applications 21:5–20

Storn R, Price K (1995) Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces

Syswerda G (1989) Uniform crossover in genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms

Vasile M, Minisci E, Locatelli M (2011) An inflationary differential evolution algorithm for space trajectory optimization. IEEE Transactions on Evolutionary Computation 15(2):267–281

Whitley LD, Garrett D, Watson JP (2003) Quad search and hybrid genetic algorithms. In: Proceedings of the Genetics and Evolutionary Computation Conference (GECCO-2003), Springer, Chicago, IL, USA, vol 2724, pp 1468–1480

Yang RL (2000) Convergence of the simulated annealing algorithm for continuous global optimization. Journal of Optimization Theory and Applications 104(3):691–716